Rapport de stage :

Étude et mise en place de signatures numériques déléguées

Auteur :

Léo Letouzey — Master Cryptologie et Sécurité Informatique, Université de Bordeaux 1

Encadrants :

Amandine Jambert — Doctorante, Orange Labs Caen

Michel Milhau — Ingénieur R&D en sécurité des services et des réseaux, Expert Sénior, Orange Labs Caen

Gilles Zémor — Professeur de Mathématiques, Université de Bordeaux 1

Stage effectué du 1er avril au 29 août 2008

Remerciements

Je souhaite ici remercier Fabrice Clerc, responsable du laboratoire MAPS /NSS/SPR pour m'avoir accueilli. Je remercie également Amandine Jambert et Michel Milhau mes responsables pour m'avoir permis d'effectuer mon stage dans les meilleures conditions possibles. Je tiens à remercier Sébastien Canard et Fabien Laguillaumie avec qui j'ai eu l'occasion de travailler.

Enfin je remercie l'ensemble de l'équipe (au sens large) du laboratoire MAPS/NSS pour m'avoir permis de m'intégrer rapidement notamment grâce aux pauses café de 9 heure. Pour ne citer qu'eux, je remercie donc Akpedje, Deborah, Bienvenu, Lian, Pascal, Walid (apprenti et stagiaires) ainsi que Amandine (encore), Cécile, Diala, Iwen, Nizar (thésards) et Thierry (prestataire).

Je remercie à Bordeaux toute l'équipe du Master CSI, notamment M^{me} Bachoc et M. Zémor responsables du master, pour ces deux années d'enseignements ansi que M. Fleury responsable des stages.

Table des matières

Introduction

La protection des contenus à toujours était quelque chose de très important. Depuis la multiplication des offres de contenus en ligne, cette protection est devenue indispensable. Cependant ces protections peuvent parfois être contraignantes notamment lorsque l'on souhaite échanger avec une autre personne un contenu protégé.

Mon stage s'inscrit dans la continuité d'un projet plus ancien. Il s'agit d'étudier, d'améliorer et d'implémenter une solution de signatures déléguées pour une application à la protection de contenu dans un contexte familial.

Je présenterai rapidement l'entreprise et le laboratoire qui m'ont accueilli avant de présenter les bases et les différents objectifs de mon stage. J'exposerai ensuite l'ensemble des travaux effectués avant de conclure.

Chapitre 1

L'Entreprise

1.1 France Telecom

1.1.1 Un peu d'histoire

En 1941, le ministère des PTT (Poste, Télégraphe et Télécommunications) crée la Direction Générale des Télécommunications (DGT) puis en 1944 le Centre nationale d'Étude des Télécommunication (CNET) ancêtre de France Telecom Recherche et Développement. En 1988 la DGT devient France Telecom et obtient le statut d'exploitant de droit public en 1991. En 1996 France Telecom devient une Société anonyme dont le capital est ouvert au public en 1997. En 2000, France Telecom rachète une grande partie de Orange. En 2003 après avoir complètement acquis Orange, le Groupe réunit ses activités mobile (Itineris, OLA et Mobicarte) dans une filiale nommée Orange SA.

Fin 2004 l'état revend une partie de ses actions, possédant moins de 50% du capital, France Telecom devient une entreprise privée. Depuis juin 2006, France Telecom utilise Orange comme marque commerciale pour vendre l'ensemble de ses produits (internet, téléphonie...) dans le monde.

1.1.2 Aujourd'hui

Fin juin 2008, le groupe compte plus de 174 millions de clients à travers le monde (plus des 2 tiers sous la marque Orange). Ainsi au 30 juin 2008 le groupe comptait :

- 113 millions de clients mobile
- plus de 12 millions de clients internet haut débit en ADSL

En 2007, le chiffre d'affaire du groupe était de $52,9$ milliards d'euros ($26,3$ milliards pour la première moitié 2008). Au 31 mars le groupe comptait près

de 190000 employés répartis dans le monde. Fin 2007, il était le premier fournisseur européen d'accès internet haut débit, troisième opérateur mobile européen, premier opérateur européen également pour la voix et la télévision sur IP (deuxième mondial pour la télévision). Grâce à sa Marque Orange Business Services, le Groupe France Telecom est parmi les principaux fournisseurs de services aux grandes entreprises.

Enfin, aujourd'hui le Groupe est présent dans plus de 220 pays et territoires.

1.2 France Telecom R&D - Orange Labs

1.2.1 Rapide présentation

La division R&D a pour principales missions :
- de développer des produits et services pour le groupe, en respectant la qualité de service ;
- de dégager de nouvelles sources de croissance ;
- d'anticiper les révolutions technologiques et d'usage ;
- d'imaginer dès maintenant les solutions du futur.

D'un point de vue stratégique, la division R&D est un avantage important pour le groupe France Telecom pour inventer de nouvelles générations de services, pour orienter l'innovation du secteur des télécommunications mais également pour anticiper les grandes ruptures technologiques. Depuis 2007, France Telecom R&D est connue sous le nom d'Orange Labs.

En 2007, France Telecom a reçu d'AT Kearney le titre de *Best Innovator* dans la catégorie Innovation organisationnelle et partenariat marketing/R&D/Réseaux SI.

1.2.2 Orange Labs en chiffres

La Division Orange Labs compte aujourd'hui 17 sites dans 8 pays différents (France, Pologne, Royaume-Uni, Japon, Corée, Chine, États Unis et Égypte depuis le premier janvier 2008). Elle emploie plus de 3800 chercheurs ingénieurs ainsi qu'un grand nombre de doctorants et post-doctorants. France Telecom y a investi $1,7\%$ du chiffre d'affaire en 2007 soit 894 millions d'euros. Les Oranges Labs sont à l'origine de plus de 8500 dépôts de brevets avec une moyenne de 400 nouveaux brevets par an.

1.2.3 Laboratoire MAPS/NSS

Les Laboratoires Orange Labs sont divisés en Centre de Recherche et Développement (CRD). Le CRD MAPS (pour Middleware et Plates-formes Avancées), composé de 7 laboratoires répartis sur 5 sites, emploie plus de 650 personnes. Il est dirigé par M^{me} Colaitis. Il a pour buts de :

- développer les composants *middleware* (primitives), les plates-formes de services et les terminaux de l'opérateur intégré ;
- définir un urbanisme de plates-formes de services interopérables, permettant une ouverture à des tiers contrôlée par l'opérateur, mutualisant les fonctions et données communes, et en cohérence avec le SI et le cœur de réseau ;
- définir une architecture technique modulable et les outils de création de services associés pour diminuer le *time to market* des services et les coûts ;
- déployer les services de communication, de production et de consommation de contenus sur tous les terminaux et contribuer à leur déploiement.

Ansi l'objectif principal du CRD MAPS est de mettre le client au centre de son propre univers de communications.

Le laboratoire Sécurité des Services et des Réseaux (NSS), dirigé par Thierry Baritaud, fait partie du CRD MAPS, c'est là que j'ai effectué mon stage. Le Laboratoire NSS a pour mission de définir et développer les solutions de sécurité permettant de protéger des malveillances les systèmes, réseaux et ressources de l'opérateur et de ses clients. Il doit aussi développer une offre de services de confiance directement perçue par le client, et sécuriser l'accès aux services et leur exécution. Le laboratoire développe le patrimoine intellectuel du groupe en matière de sécurité (cryptographie notamment), produit des briques de confiance (composants logiciels, algorithmes, plates-formes,...) et développe une expertise technique lui permettant de contribuer à l'aspect sécurité des projets de croissance du Groupe.

Chapitre 2

Présentation du stage

2.1 Sujet, Contexte

Le principe des signatures est de garantir l'origine et l'intégrité du contenu qu'elles accompagnent. En règle générale, on souhaite que ces signatures ne puissent être construites que par une seule personne, celle qui est à l'origine du contenu. Dans la pratique, le Signataire signe le contenu en chiffrant un haché à l'aide de sa clé privée. Tout le monde peut ensuite vérifier l'origine et la consistance du contenu en vérifiant la signature et en comparant le résultat avec le haché du contenu.

Cependant, il peut arriver qu'une personne, autre que le Signataire, ait besoin d'effectuer des modifications (voir même des suppressions) de certaines parties du contenu ; tout en gardant une signature valide pour ce contenu modifié. Bien sur il va de soit que le signataire d'origine est conscient de cette possibilité. La solution triviale consiste à demander une nouvelle signature sur le contenu modifié au Signataire, mais il est possible que cette solution soit trop contraignante, voir même impossible. Par exemple si le Signataire est injoignable ou si l'on est amené à modifier fréquemment le contenu et que chaque demande de signature est trop coûteuse...

Les **Signatures déléguées**[1] [3] [2] vont dans ce sens. Elles permettent au Signataire de choisir une (ou plusieurs) personnes (Rédacteurs) qui seront autorisées à effectuer des modifications sur certains morceaux choisis du contenu. Les Rédacteurs ont alors la possibilité de modifier le contenu tout en gardant une signature valide sur le nouveau contenu.

Cela peut être utile par exemple pour des données médicales. Toutes les informations sur un patient sont contenues dans le document original, seulement un médecin n'a pas forcément besoin de toutes ces informations. Le document sera donc modifié pour ne laisser que les informations indispen-

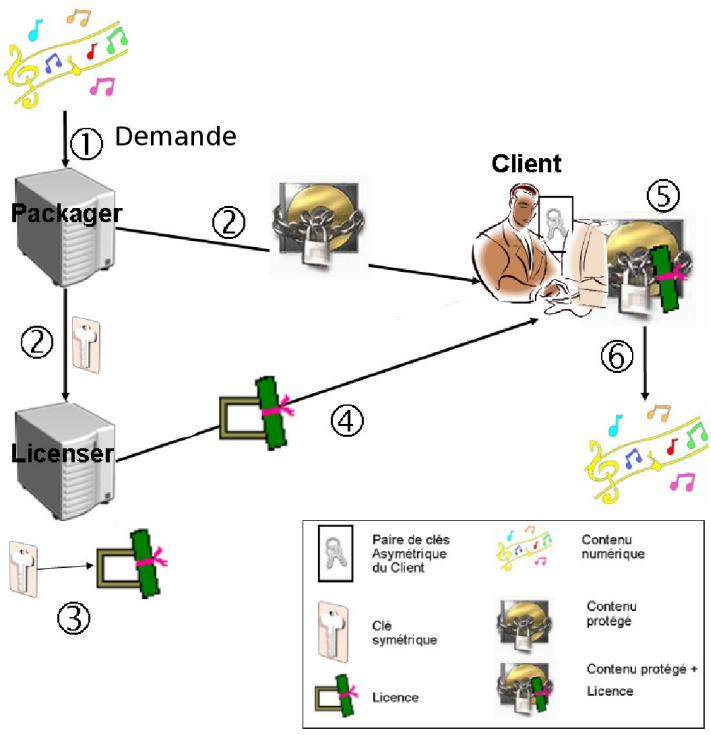
sables (ou non sensibles) et le médecin pourra toujours vérifier l'authenticité du document avec la signature qui lui sera fournie. Un autre exemple est la distribution de contenu multimédia (un film...), certaines parties peuvent être censurées, certains sous-titres modifiés pour s'adapter au mieux à l'âge du public.

2.1.1 Contexte du stage

Le but du stage était d'étudier une solution de signature déléguée applicable dans le cadre d'une distribution de contenus protégés dans un contexte familial. La gestion des droits est basée sur le modèle des DRM.

2.1.2 Modèle DRM

Le système DRM (*Digital Rights Management*, ou Gestion des Droits Numériques (GDN) en français) permet de protéger les contenus numériques de toutes sortes (musique, vidéo, image, texte...) de façon à empêcher tout usage non autorisé.



Il met en relation trois personnes :
- le Packager ;
- le Serveur de Licences (SL) ;
- l'Agent DRM (DRMA).

Le Rôle du Packager est de chiffrer le contenu choisi par un utilisateur avec une clé symétrique appelée CEK (Content Extraction Key), de fournir le contenu chiffré à l'utilisateur et de donner la CEK au SL. La CEK change en général pour chaque contenu. Le SL fournit les licences qui vont permettre à l'utilisateur de profiter du contenu. Les licences sont construites de la façon suivante :

$$L = [Id_L, Id_c, m, CEK_c, S]$$

où :
- Id_L et Id_c sont les identifiants de la licence et du contenu ;
- m correspond aux droits accordés par la licence (lecture, transfert...) ;
- CEK_c est la CEK chiffrée à l'aide de la clé publique de l'utilisateur ;
- S est une signature sur les champs précédents (signés avec la clé privée du SL).

Enfin le DRMA se charge de déchiffrer le contenu à l'aide des informations contenues dans la licence. Le contenu en clair ne doit pas sortir du DRMA afin de ne pas se retrouver sans protection. On peut considérer, pour faciliter la compréhension, que le Packager et le SL sont une seule entité.

Fonctionnement :

1. L'Utilisateur demande un contenu au SL (au Packager).
2. Le SL chiffre le contenu demandé avec une CEK.
3. Le SL chiffre la CEK avec la clé publique de l'utilisateur et construit la licence qui ira avec le contenu chiffré.
4. Le DRMA de l'utilisateur reçoit le contenu chiffré et la licence correspondante.
5. Le DRMA déchiffre la CEK puis le contenu et vérifie les informations de la licence.
6. L'utilisateur peut profiter du contenu qu'il vient d'acquérir.

2.1.3 Application à notre contexte familial

Dans notre cas, on souhaite pouvoir transférer un contenu d'une personne à l'autre. Tel quel, le modèle DRM empêche de le faire. En effet chaque licence est personnalisée pour un unique utilisateur. L'utilisateur B ne saura

pas déchiffrer la CEK contenue dans une licence destinée à un utilisateur A, il n'aura donc pas accès au contenu.

On utilise donc les signatures déléguées pour permettre à un des utilisateurs de faire des modifications dans la licence pour qu'elle puisse être utilisée par un autre utilisateur. D'un point de vue pratique, l'utilisateur A reçoit une licence L que le SL l'autorise à modifier (A est un Rédacteur). A déchiffre la CEK grâce à sa clé privée puis la chiffre avec la clé publique de B. Il modifie alors la Licence avec cette nouvelle CEK chiffrée et transmet à B le contenu chiffré et la nouvelle licence. B peut maintenant utiliser cette nouvelle licence comme n'importe quelle licence et profiter du contenu puisqu'il est maintenant en mesure de déchiffrer la CEK.

2.2 État de l'art

Il existe deux approches bien distinctes pour les signatures déléguées. Si la finalité reste la même, obtenir une signature valide sans que le Signataire l'ait explicitement calculée, les motivations peuvent être différentes. On peut par exemple vouloir construire une signature à partir d'éléments déjà existants ou au contraire obtenir une signature sur un message inédit et inconnu du Signataire au moment de la signature initiale. Dans ce dernier cas, on peut encore distinguer deux objectifs, soit on souhaite uniquement supprimer certaines parties du message d'origine, soit on souhaite en modifier certains morceaux.

2.2.1 Signatures Transitives

Les signatures transitives[12] [11] sont utiles lorsque l'on souhaite faire des combinaisons de messages signés et être capable d'obtenir une signature sur ce message sans avoir besoin du Signataire.

Le premier exemple que l'on peut donner est basé sur la signature RSA[1]. Si l'on dispose de deux signatures sur deux messages, m_1 et m_2, $\sigma(m_1)$ et $\sigma(m_2)$, n'importe qui est en mesure de calculer $\sigma(m_1 m_2)$. En effet :

$$\left.\begin{array}{l}\sigma(m_1) = {m_1}^d \bmod n \\ \sigma(m_2) = {m_2}^d \bmod n\end{array}\right\} \Rightarrow \sigma(m_1)\sigma(m_2) = {m_1}^d {m_2}^d = (m_1 m_2)^d = \sigma(m_1 m_2)$$

Les signatures transitives sont également adaptées aux graphes. Comme présenté dans [7] par Rivest. Si l'on dispose des signatures sur les segments (AB) et (BC), il peut être intéressant de pouvoir obtenir la signature sur (AC) sans intervention du Signataire. Ainsi si le Signataire veut ajouter un

[1] cette propriété est également utile pour les signatures aveugles.

point au graphe, il n'a besoin de signer qu'un seul segment rattachant ce nouveau point au reste du graphe, les signatures sur les autres segments créés pouvant être déduites de cette seule signature.

Supposons que l'on dispose des signatures sur les segments (AB) et (BC) on sera alors en mesure de construire la signature sur le segment (AC).

$$\begin{array}{ccc} A & \longleftrightarrow & B \\ & \ddots & \updownarrow \\ & & C \end{array}$$

Les signatures transitives peuvent être très pratiques dans certains cas, notamment dans l'exemple précédent si le Signataire est amené à ajouter fréquemment de nouveaux points au graphe. Il n'est plus forcé de signer l'intégralité des nouveaux segments. Il y a cependant un certains nombres de points faibles qui rendent cette méthode inutilisable dans notre cas. Tout d'abord, le Signataire doit signer tous les nouveaux éléments, rien ne peut être ajouté par une autre personne. Ensuite, n'importe qui disposant de $\sigma(A, B)$ et de $\sigma(B, C)$ peut calculer $\sigma(A, C)$, cela peut être gênant si l'on souhaite garder un minimum de contrôle sur les signatures.

2.2.2 Fonction de Hachage Caméléon

Les fonctions de hachage caméléon (*Chameleon Hash*, CH dans la suite) ont été présentés pour la première fois dans [5]. Le CH est une fonction de hachage qui a une particularité très intéressante. Elle permet en effet à celui qui calcule la première fois de choisir une personne qui aura la possibilité de construire des collisions de façon à ce que les messages qu'elle choisis aient le même CH que le message d'origine.

Fonctionnement

Préliminaires

Le Signataire choisit un rédacteur qu'il autorise à effectuer des modifications sur les messages qu'il fournit. Il récupère la clé public y du Rédacteur. y est construite de la manière suivante : soient p et q deux nombres premiers tels que : $p = uq + 1$, g d'ordre q et $x \in [0 \dots q-1]$. Alors : $y = g^x \bmod p$.

Hachage

Le Signataire choisit $\alpha, \beta \in [0 \dots q-1]$ et calcule le paramètre $e = H(m, \alpha)$. Où $H(.,.)$ est une fonction de hachage standard et m le message à hacher. Finalement, il calcule le CH de m :

$$CH(m, \alpha, \beta) = \alpha - (y^e g^\beta \bmod p) \bmod q$$

Dans la suite du rapport, on notera R le couple (α, β) qui sera appelé le paramètre du CH. Si ce paramètre est évident et qu'il n'est pas utile dans le contexte, on notera juste $CH(m)$ pour $CH(m, \alpha, \beta)$.

Collision

On a : $C = CH(m, \alpha, \beta)$. Le but de la construction de collision est de trouver α' et β' tels que :

$$CH(m, \alpha, \beta) = C = CH(m', \alpha', \beta')$$

Celui qui veut faire cette modification doit disposer de la clé secrète x. Il commence par choisir un $k \in [1, q+1]$. Il calcule ensuite :

$$\alpha' = C + (g^k \bmod p) \bmod q$$

$$e' = H(m', \alpha')$$

$$\beta' = k - e'x \bmod q$$

On a alors :

$$CH(m', \alpha', \beta') = C$$

Démonstration

On a :

$$\begin{aligned} CH(m', \alpha', \beta') &= \alpha' - (y^{e'} g^{\beta'} \bmod p) \bmod q \\ &= C + (g^k \bmod p) \bmod q - (g^{xe'} g^{k-xe'} \bmod p) \bmod q \\ &= C + (g^k \bmod p) \bmod q - (g^k \bmod p) \bmod q \\ &= C \\ &= CH(m, \alpha, \beta) \end{aligned}$$

□

Si quelqu'un souhaite modifier la licence et qu'il dispose de la clé secrète x, il lui suffit de calculer une collision pour le CH concerné et de mettre à jour le paramètre a la fin de la signature. La valeur de ce qui est signé ne changeant pas, la signature sera toujours valide après la modification.

Pour vérifier la signature de la licence, il faut commencer par calculer, à l'aide des paramètres fournis, les CH puis de construire le nouveau message comme dans l'étape de signature et finalement de vérifier que la signature est correcte.

2.2.3 Application des fonctions de hachage caméléon pour la signature d'un message

Utiliser les CH directement sur le message complet n'est pas une bonne idée. En effet, cela permet au Rédacteur de modifier entièrement le message et de prétendre que celui-ci vient du Signataire. Le Rédacteur peut donc créer les messages qu'il souhaite et faire en sorte qu'ils soient signés par le Signataire. Cela revient presque à fournir au Rédacteur la clé privée du Signataire.

En règle générale, certaines parties du message d'origine n'ont pas à être modifiées. On peut penser par exemple à la date de naissance d'un patient dans un dossier médical ou à l'identifiant de la licence dans le cas des DRM... Le Signataire doit donc empêcher les modifications sur certains morceaux du message tout en les autorisant sur d'autres.

Pour cela il découpe le message en différents morceaux qu'il peut décider comme étant modifiables ou non. Ainsi au lieu de signer le message tel quel, il signera la concaténation des morceaux non modifiables et des CH des morceaux modifiables. Il fournira les paramètres des différents Cameleon Hash pour permettre à la fois les modifications par les Rédacteurs et la vérification pour tout le monde.

Le Rédacteur, pour modifier le message, n'a besoin que de calculer les collisions de CH et de mettre à jour les paramètres à la fin de la signature.

Remarques

Le Signataire peut choisir des Rédacteurs différents pour chaque partie modifiable du message. Il peut choisir *qui* modifie *quoi* dans le message.

Fonctionnement

Soit $m = \{m_1, m_2 \ldots, m_n\}$ le message à signer. Soit $K \subseteq [0, \ldots, n]$ l'ensemble des indices des morceaux modifiables. Enfin CH est notre fonction de hachage caméléon.

Signature

Le SL commence par construire :

$$m' = \{m_i{}'\} \text{ où : } m_i{}' = \begin{cases} m_i & \text{si } i \notin K \\ CH(m_i, r_i) & \text{si } i \in K.\ r_i \text{ choisi aléatoirement} \end{cases}$$

Puis il signe m' avec sa clé privée :

$$\sigma(m') = Signe_{SL}(m')$$

La signature finale est la concaténation de la signature obtenue juste avant et des différents paramètres des CH :

$$S = [\sigma||r_{i_1}||\dots||r_{i_k}]$$

où :

$$[i_1, \dots, i_k] = K$$

Modification

Le principe est le même que celui déjà évoqué lors de la présentation des collisions du CH. On calcule le nouveau paramètre r'_i que l'on substitue à l'ancien dans la signature S originale.

On obtient alors :

$$S = [\sigma||r_{i_1}||\dots||r'_{i_j}||\dots||r_{i_k}]$$

Bien sur il est possible de modifier plusieurs morceaux en même temps.

Vérification

Le vérifieur reçoit un message $m = \{m_1, m_2 \dots, m_n\}$ et la signature qui l'accompagne, $S = [\sigma||r_{i_1}||\dots||r_{i_k}]$.

Il commence par construire le message $m' = \{m_i{}'\}$ tel que :

$$m_i{}' = \begin{cases} m_i & \text{si } i \notin K \\ CH(m_i, r_i) & \text{si } i \in K \end{cases}$$

Il vérifie ensuite que $\sigma(m')$ est correcte.

Conclusion

Contrairement aux signatures transitive, il est possible grâce au CH d'introduire dans le message des morceaux que le SL n'a jamais vus. Cela offre donc une plus grande liberté d'action à ceux qui effectuent les modifications. D'autre part, les modifications ne peuvent être effectuées que par la personne qui détient la clé secrète x. Alors que dans le cas des signatures transitives, n'importe qui pouvait construire de nouvelles signatures.

Cependant avec ce modèle, le Rédacteur peut, dès que le message lui est transmis effectuer des modifications. Le SL peut vouloir ne pas autoriser les modifications dans un premier temps pour dans un deuxième temps permettre au Rédacteurs de changer le message. C'est là qu'apparaît la solution présenté par Sébastien Canard, Fabien Laguillaumie et Michel Milhau sur laquelle j'ai travaillé.

Note :

Dans la suite du rapport, lorsque je fais référence à «la partie fixe de la signature», il s'agit de l'ensemble des morceaux non modifiables du message. Lorsque je parles de «la fin de la signature», je fais référence aux différents paramètres ajoutés à la signature pour permettre sa vérification.

2.2.4 La solution existante

La solution sur laquelle j'ai travaillé, nommée **Trapdoor Sanitizable Signature** [6] (TSS dans la suite), est une amélioration des techniques présentées dans les articles précédents.

Le but des TSS est de permettre au Signataire de choisir non seulement qui pourra effectuer des modifications mais aussi de pouvoir décider quand il pourra les faire. Jusque là, le signataire choisissait un Rédacteur et utilisait sa clé publique dans le CH. Le Rédacteur pouvait donc aussitôt effectuer des modifications. Ici ce pouvoir n'est donné que lorsque le Signataire le décide. Le principe reste le même, c'est à dire remplacer les morceaux modifiables par leur haché avant de signer le message. Seulement maintenant, pour pouvoir changer la valeur de ces morceaux, le Rédacteur doit connaître la valeur d'une **trappe** qui lui permettra de calculer les collisions souhaitées. Cette trappe sans laquelle il ne peut rien lui est fournie par le Signataire. Ainsi, le Signataire peut distribuer les messages signés sans se soucier d'autoriser ou non les modifications et lorsque le Rédacteur voudra faire les premières modifications, il demandera la trappe au Signataire.

La construction du CH a un peu changée par rapport au modèle précédent.

fonctionnement

Soient (p, q, w) et (n, v) une paire de clé de type RSA, $n = pq$.(i.e. w est l'inverse de v modulo $\varphi(n)$).

Construction de la trappe

À partir de l'identité du destinataire du message, le Signataire calcule

$$J = EMSA - PSS(Id)$$

et construit la trappe :

$$\tau = J^w \bmod n$$

Hachage

Soit m le message à hacher. Le Signataire choisie un paramètre r aléatoire mod n et calcule

$$h = J^{H(m)} r^v \bmod n$$

Bien sur H est une fonction de hachage standard.

Collision

À l'aide de la trappe, du message d'origine m et du nouveau message m', le Rédacteur peut construire le nouveau paramètre r' vérifiant le nouveau message.

$$r' = r\tau^{H(m)-h(m')} \bmod n$$

Vérification

À partir du message, du paramètre r et de l'Id du destinataire, n'importe quel utilisateur peut vérifier la valeur du haché. L'utilisateur commence par calculer

$$J = EMSA - PSS(Id)$$

Puis vérifie si

$$h \stackrel{?}{=} J^{H(m)} r^v \bmod n$$

Démonstration

Soient les couples (message, paramètre) (m, r) et (m', r') construits de façon à ce que : $CH(m, r) = CH(m', r')$.
On a donc :

$$\begin{aligned} CH(m', r') &= J^{H(m')} r'^v \bmod n \\ &= J^{H(m')} r^v \tau^{v(H(m)-H(m'))} \bmod n \\ &= J^{H(m')} r^v J^{vw(H(m)-H(m'))} \bmod n \\ &= J^{H(m')} r^v J^{H(m)-H(m')} \bmod n \\ &= J^{H(m)} r^v \bmod n \\ &= CH(m, r) \bmod n \end{aligned}$$

□

Remarque :

Le choix de $EMSA - PSS$ [10] est arbitraire est peut être remplacé par n'importe quelle fonction de padding classique.

Conclusion

La principale amélioration apportée par ce schéma est que le Signataire peut décider a quel moment il autorise le Rédacteur à faire des modifications (contrairement au schéma classique présenté juste avant).

Cela peut être utile dans notre modèle de DRM. Un utilisateur peut demander un contenu au SL qui lui fournie avec la licence correspondante. Si après cela l'utilisateur souhaite devenir Rédacteur, il faut qu'il demande au SL la trappe associée à la licence. Le SL n'a pas besoin de reconstruire toute la licence et fournit (s'il est d'accord) la trappe τ.

2.2.5 Conclusion

Nous venons de voir plusieurs versions de signatures déléguées. La première méthode (signatures transitives) ne correspond pas à nos besoin puisqu'elle ne permet pas d'ajouter des informations inconnues du SL. La méthode des Trapdoor Sanitizable Signatures est la base des travaux effectués au cours de mon stage. Le but étant de l'améliorer et de l'adapter à l'exemple des DRM. Dans le chapitre suivant, nous verrons quelles ont été les modifications apportées est comment elles ont été mises en place.

Chapitre 3

TSS, objectifs et solutions apportées

3.1 Objectifs

Malgré tous les avantages apportés par la mise en place des TSS dans notre modèle, il reste certains problèmes.

3.1.1 Mixage de messages

La premier objectif est d'empêcher la fabrication de messages non légitimes. En effet, il existe un moyen très simple de construire un message même si l'on ne fait pas parti des Rédacteurs.

Après plusieurs modifications d'un même message, si l'on dispose d'au moins deux versions du message, il est possible de *mixer* les différentes versions pour obtenir un nouveau message inédit pourtant accompagné d'une signature correcte.

Exemple

Prenons un exemple simple ou le message m n'est composé que de deux morceaux modifiables. On dispose de deux versions de ce message, m_1 et m_2.

$$m_1 = [A, B] \text{ et } m_2 = [C, D]$$

Accompagnés de leur signature S_1 et S_2.

$$S_1 = [Sign_{SL}(CH(A), CH(B)), r_A, r_B] \text{ et } S_2 = [Sign_{SL}(CH(C), CH(D)), r_C, r_D]$$

N'importe qui peut maintenant construire le message $m_3 = [A, D]$ accompagné de la signature $S_3 = [Sign_{SL}(CH(A), CH(B)), r_A, r_D]$. La vérification se passera sans encombre puisque l'on a bien :

$$CH(B, r_B) = CH(D, r_D)$$

3.1.2 Contrôler les modifications

Le second objectif est d'autoriser les modifications tout en gardant quand même le contrôle sur les valeurs qui sont changées. Ainsi on peut souhaiter permettre les modifications d'un champ uniquement dans un ensemble bien précis de valeurs connues. Par Exemple, si un des morceaux modifiables d'un message concerne une date ou un nom, il est préférable qu'après modification on conserve une certaine consistance du message. Il ne faudrait pas par exemple qu'un Rédacteur remplisse un champ contenant une date par des lettres, un nombre négatif, etc. On veut ainsi laisser la liberté au Rédacteur de changer le message tout en garantissant la bonne formation du message.

3.1.3 Ajout d'un compteur

Pour l'instant un Rédacteur peut fournir autant de versions du message qu'il le souhaite. Le Signataire peut souhaiter limiter ce nombre. Il faut bien sur que le système soit statique puisqu'une fois la Trapdoor échangée, le Signataire et le Rédacteur ne sont plus supposés communiquer. Il faut donc contraindre le Rédacteur sans intervention du Signataire. Il faut également pouvoir détecter les éventuelles fraudes.

La mise en place du compteur soulève un nouveau problème. Si on se rend compte que le nombre de versions d'un certain message est dépassé, le Rédacteur peut prétendre qu'il n'a pas fraudé et affirmer que les versions en surnombre proviennent du Signataire. En effet, comme il suffit de connaître la trapdoor pour pouvoir modifier un message, le Signataire peut lui aussi construire de nouvelles versions du message. Il n'est pas possible de faire la distinction sur l'origine de telle ou telle version du message. Ainsi même si le Signataire découvre trop de versions d'un même message, il ne lui sera pas possible d'accuser le Rédacteur. Il faut donc trouver un moyen de déterminer l'origine des différents messages.

3.2 Transitivité des signatures

Le problème de «*mixage*» de messages, comme expliqué précédemment, peut être utilisé par un utilisateur pour obtenir de nouveaux messages issus ni du Signataire ni d'un Rédacteur. L'idée pour remédier à ce problème est plus une astuce qui marche dans notre application qu'une véritable solution. Au lieu d'avoir un CH par morceaux modifiables, ce qui ouvre la porte aux combinaisons de messages, il suffit de n'utiliser qu'un seul Cameleon Hash pour l'ensemble des morceaux modifiables.

Pour cela, il faut concaténer les parties modifiables et utiliser le CH sur le message ainsi obtenu. On a donc un message identique à ce qu'il y avait avant. Cependant la construction de la signature a légèrement changé. Si par exemple on a :

$$m = [m_1, m_2, m_3]$$

avec m_2 et m_3 modifiables, la signature ressemblera a :

$$S = [Sign_{SL}(m_1, CH(m_2||m_3)), r]$$

Si un utilisateur B récupère une licence qui ne lui est pas destinée (celle de l'utilisateur A), il peut toujours l'utiliser pour modifier sa propre licence. Mais dans ce cas, soit il utilise dans la licence la CEK_{cb} de sa licence et à ce moment là la vérification du CH échoue, soit il utilise la CEK_{ca} de la licence qu'il a récupérée et dans ce cas, la vérification du CH s'effectue correctement mais le DRMA n'est plus en mesure de déchiffrer la CEK puisqu'elle a été chiffrée avec la clé publique de A. Le mixage des licences est donc devenu inefficace.

On peut aussi voir que cette méthode permet de réduire la taille de la signature. Il n'y a plus qu'un seul paramètre de CH à ajouter pour permettre la vérification de celle-ci au lieu de deux précédemment. Cependant, le Signataire ne peut plus choisir plusieurs Rédacteurs différents pour chaque morceau et le Rédacteur à maintenant accès à toutes les parties modifiables.

3.3 Contrôler les modifications

Permettre à quelqu'un d'autre de faire certaines modifications est une bonne chose dans certains cas, il faut cependant veiller à ce que ces modifications soient cohérentes avec le reste du message. Il faut donc trouver une solution pour contraindre le Rédacteur à mettre des valeurs qui ont un sens lorsqu'il effectue des modifications. On se placera dans le cas où le Signataire choisit l'ensemble des valeurs possibles pour un morceau modifiable du message (par exemple une date, un nom dans une liste, un ensemble de droits...).

La solution triviale consiste à ajouter dans la partie fixe de la signature du message l'ensemble des valeurs que peut prendre le morceau du message. Lors de la vérification il suffit de regarder si la valeur en question est bien dans l'ensemble signé. Cette méthode n'est bien sûr pas envisageable dès que cet ensemble est un peu grand puisqu'il faut alors ajouter à la licence toutes les valeurs de l'ensemble en clair pour que la vérification soit possible. La signature aurait alors une taille impraticable.

3.3.1 Accumulateurs

Les **Accumulateurs**[9] (*Hash Accumulator*) sont des fonctions permettant de vérifier qu'un message est bien dans un ensemble prédéfini. Ils fonctionnent à l'aide du triplet (message, témoin, accumulateur). Ainsi chaque message contenu dans l'ensemble accumulé, dispose de son propre témoin et inversement, il n'est pas possible de trouver un témoin pour un message qui n'est pas dans l'ensemble.

Fonctionnement des Accumulateurs

1. **Préliminaires :** Soit $Hacc$ la fonction d'accumulateur. Soit $N = pq$ et $p = 2p'+1$, $q = 2q'+1$ avec p, p', q, q' de grands nombres premiers (avec p et q de même ordre de grandeur). Soit x un générateur de $\mathbf{Z}_N$. Soit H^* une fonction de hachage standard. Enfin soit $M = \{m_1, \ldots, m_k\}$ l'ensemble des messages à accumuler.
2. **Hachage :** On calcule tout d'abord

   $$z = Hacc(M) = x^{\prod_{i \leq k} H^*(m_i)} \bmod N$$

   puis pour prouver que le message m_j est bien dans l'ensemble M, on calcule le témoin

   $$z_j = Hacc(m_j) = x^{\prod_{i \neq j} H^*(m_i)} \bmod N$$

   Au final, on donne z, z_j et m_j.
3. **Vérification :** Pour vérifier que le message $m_j \in M$, il suffit de vérifier que l'on a bien :

   $$z \stackrel{?}{=} z_j^{H^*(m_j)} \bmod N$$

Remarque sur les accumulateurs :
Il suffit de connaître l'ensemble M pour construire le témoin (z_j) qui accompagne le message (m_j). Il est aussi très simple de vérifier si un ensemble M' correspond à l'ensemble accumulé puisqu'il suffit de vérifier si :

$$Hacc(M) \stackrel{?}{=} Hacc(M')$$

Ainsi même si l'ensemble M n'est pas donné, il est possible de le supposer si le contexte le permet (jours de la semaine, du mois, liste de nom...) et de vérifier si il correspond à l'ensemble initial. Dans notre application cela ne sera pas gênant et on utilisera l'accumulateur tel quel. Cependant il est

possible de remédier à ce problème en ajoutant à l'ensemble M une valeur quelconque. Ainsi, il devient impossible de deviner l'ensemble M en testant $Hacc(M) \stackrel{?}{=} Hacc(M')$, seuls ceux connaissant cette valeur supplémentaire seront en mesure de calculer les valeurs des témoins accompagnant les messages.

La quantité de calculs est très inégale entre le hachage et la vérification. La vérification ne consistant qu'en une puissance modulaire, elle ne pose pas de problème. Cependant si M est assez grand et que le Rédacteur n'a qu'une capacité de calcul limitée (s'il s'agit d'un terminal mobile par exemple), il faut pouvoir contourner le problème. Une solution serait de faire calculer l'ensemble des z_j au Signataire (qui, on peut le supposer, a une capacité de calcul suffisante), puis le Signataire fournirait en même temps que le message et la trapdoor, l'ensemble des z_j au Rédacteur. Le Rédacteur n'aurait alors plus aucun calcul à faire. Il faut dans ce cas que le Rédacteur ait une capacité de stockage suffisante car il doit pouvoir stocker l'ensemble M et l'ensemble des z_j.

D'autre part cette méthode peut engendrer des calculs inutiles, il est très probable que le Rédacteur n'ait pas besoin de connaître tous les z_j s'il ne souhaite pas par exemple utiliser toutes les valeurs possibles pour ce morceau du message. Enfin le volume des communications est plus important ce qui peut aussi être problématique.

3.3.2 conclusion

La mise en place des accumulateurs est une solution qui permet de contrôler les modifications sans surcharger la licence. Elle est en plus adaptable aux capacités de calcul du terminal qui l'utilise donc applicable sur toute sorte de supports (mobile, serveur...). Le surplus de calculs est minime voir même nul si le Signataire se charge des pré calculs.

3.4 Ajout d'un compteur et identification du Rédacteur

Pour le moment le modèle que l'on a ne propose pas de solution permettant de limiter le nombre de version d'un même message qui peuvent être produites. Ainsi, s'il le souhaite, le Rédacteur peut fournir autant de version du message qu'il le souhaite. Cela peut être contraignant dans notre cas de distribution de licences. Il faut donc trouver une solution limitant la multiplication des versions d'une même licence.

De nombreuses approches ont été envisagées pour concevoir la mise en place d'un compteur. Il a fallu définir précisément ce qu'apporterait un tel compteur. Il doit fonctionner sans l'intervention du Signataire (puisqu'une fois la trapdoor échangée, le Rédacteur et le Signataire ne sont plus sensés communiquer), l'utilisateur qui reçoit la version du message ne doit pas non plus être en contact avec le Signataire. Ce doit être le Signataire qui fixe le nombre de versions d'une licence que le Rédacteur peut produire. La méthode choisie pour contraindre le Rédacteur à respecter cette limite est de révéler une information sensible le concernant en cas de fraude (information à la portée de tous).

Cependant la mise en place de sanctions soulève un nouveau problème. Il faut s'assurer de l'origine des versions du message qui circulent. En effet, le Signataire est lui aussi en mesure de fournir des messages avec signatures valides. Ainsi le fait de voir des messages en surnombre n'implique pas nécessairement que le Rédacteur a fraudé.

On souhaite donc que le compteur ait les propriétés suivantes :

1. vérification statique (Sans intervention du Signataire) ;
2. identification de l'origine de la copie ;
3. empêcher la réutilisation d'un même compteur ou de dépasser la limite autorisée ;
4. révélation d'information(s) sur le Rédacteur en cas de fraude.

3.4.1 ZKPK

Pour assurer l'origine de chaque version du message, on utilise une version un peu modifiée du **Protocole de Schnorr** [8]. Il s'agit d'une Preuve à divulgation nulle de connaissance (*Zero Knowledge Proof of Knowledge*, ZKPK ou *Zero Knowledge Interactive Proof*, ZKIP en anglais). Pour la suite, on utilisera ZKPK pour parler de Preuve à divulgation nulle de connaissance.

Définition :
Une ZKPK est un protocole qui permet à une personne (le fournisseur de preuve) de fournir à une seconde personne (le vérifieur) la preuve qu'elle possède une certaine information. À la fin du protocole, le vérifieur est convaincu que le fournisseur de preuve est bien en possession de l'information. Il n'a cependant rien appris d'autre (notamment rien sur l'information possédée par le fournisseur de preuve).

Une ZKPK doit avoir les propriétés suivantes :

Consistance (*completness*) : Tant que le fournisseur de preuve et le vérifieur suivent le protocole, le vérifieur est convaincu par la preuve fournie.

Solidité (*soundness*) : Si un fournisseur de preuve ne dispose pas de l'information à prouver, il ne peut pas convaincre le vérifieur avec une forte probabilité.

Divulgation nulle (*zero knowledge*) : À la fin du protocole, le vérifieur n'apprend rien d'autre que le fait que le fournisseur de preuve est bien en possession de l'information. Cette proposition tient même si le vérifieur triche et ne suit pas le protocole.

La ZKPK va être utile dans notre cas pour différencier les copies de licences issues du SL et celles issues du Rédacteur. Seul celui à l'origine de la copie pourra jouer le rôle de fournisseur de preuve. Le Rédacteur ne pourra alors plus prétendre que le SL est à l'origine des licences en trop.

Protocole de Schnorr

Le protocole de Schnorr, présenté dans [8], fonctionne de la façon suivante : On note A le Fournisseur de preuve et B le vérifieur. A commence par s'*engager* sur la valeur du secret, il doit ensuite répondre correctement au *challenge* proposé par B.

Préliminaires : Soient p et q deux nombres premiers tels que : $q|(p-1)$ et $\alpha \in \mathbf{Z}_p$, α d'ordre q. p, q, α sont publiques.

étape 1 : Soit s l'information à prouver, $s \in \mathbf{Z}_q$. A calcule $v = \alpha^{-s} \bmod p$ et le donne à B (v est la clé publique de A). A choisit r aléatoirement dans $[1, \ldots, q-1]$, calcule $x = \alpha^r$ et donne x à B. (engagement)

étape 2 : B reçoit v et x. Il choisit $e \in \mathbf{Z}_q$ et l'envoie à A. (challenge)

étape 3 : A reçoit e et calcule $y = r + se \bmod q$. Il envoie y à B.

étape 4 : B reçoit y et vérifie si : $x \stackrel{?}{=} \alpha^y v^e \bmod p$. Si l'égalité est correcte, B peut être convaincu que A connaît bien l'information s.

Remarque :

Seul celui qui dispose de s peut répondre correctement au *challenge* de B (étape 2). Ainsi, si le Rédacteur est le seul à connaître s, cela suffit à garantir l'origine des copies de licence. Seulement le protocole de Schnorr est interactif (étapes 2 et 3). Comme on souhaite avoir une méthode d'identification statique, il faut contourner le problème.

Protocole statique

Le protocole de Schnorr nécessite l'intervention du vérifieur lors de l'étape 2. Son unique rôle est de choisir un e aléatoire de façon à éviter que A connaisse e au moment où il s'engage sur r (étape 1 en donnant x). Il faut

donc trouver une méthode qui permet à la fois de contraindre A sur les valeurs de r et qui permette que r reste secret pour B. Il faut ensuite trouver une façon pour A de fabriquer un aléa qu'il ne maîtrise pas et que B puisse vérifier.

L'idée est de prendre le paramètre du CH pour jouer le rôle de l'aléa e et d'utiliser r comme un compteur.

De part la construction du CH, lorsque le Rédacteur construit un collision, il obtient un paramètre R différent à chaque fois que le message change. La valeur de R est répartie uniformément dans $\mathbf{Z}_N$ sans que A puisse contrôler sa valeur. En fait, le Rédacteur peut fixer au choix soit R, soit le message à modifier, celui qui n'est pas fixé se comporte alors comme un aléa. Étant donné que A est obligé de fixer le message, on peut utiliser le paramètre du CH comme aléa pour le protocole de Schnorr.

Il faut maintenant faire en sorte que A soit forcé de fixer r avant de calculer R. De plus le nombre de r à décider dépend du nombre de version du message que peut fournir le Rédacteur.

Pour contraindre A à fixer les r avant de construire le CH, on va réutiliser les accumulateurs. Au moment où A reçoit la trapdoor de la part du Signataire, il commence par choisir autant de r_i que de versions autorisées. IL calcule ensuite les $t_i = \alpha^{r_i}$ correspondant ($\mathcal{T} = \{t_i\}$). Il calcule également $v = \alpha^s$. les r_i servent d'engagements pour A (étape 1) et v joue le rôle de clé publique. A est le seul à connaître les r_i et s. Le Signataire calcule alors $Hacc(\mathcal{T})$ et l'ajoute à la partie fixe de la signature du message il y ajoute également v. À chaque nouvelle version, le Rédacteur utilise un des r_i comme engagement et ajoute à la Signature le témoin qui prouve que le t_i correspondant est bien dans l'accumulateur.

Protocole statique :

Préliminaires : A choisit un ensemble de valeurs $\{r_i\}$ aléatoires et calcule les $t_i = \alpha^{r_i}$, il choisit un s et calcule $v = \alpha^s$. Il donne $\mathcal{T} = \{t_i\}$ et v au Signataire qui les ajoute dans la partie fixe de la signature.

étape 1 : A choisit un des t_i et calcule le paramètre, R, du CH de $M = (m_1||m_2||\ldots||t_i)$ où les m_i sont les morceaux modifiables du messages. Il calcule aussi le témoin du t_i utilisé, $Hacc(t_i)$. Enfin, il ajoute t_i et $Hacc(t_i)$ à la fin de la signature.

étape 2 : A calcule $y = r_i - Rs$, qu'il ajoute également à la fin de la signature.

Vérification : Lorsque B reçoit le message, il commence par vérifier que le CH est construit correctement, il vérifie ensuite que le t_i fourni est bien dans l'accumulateur à l'aide du témoin et de l'accumulateur. Il regarde

finalement si on a bien :

$$t_i \stackrel{?}{=} v^R \alpha^y \bmod p$$

Détection de fraude

À chaque nouvelle version d'un message, le Rédacteur est maintenant obligé de choisir un des r_i. Il peut donc fournir autant de messages qu'il a de r_i différents sans avoir à utiliser le même r_i deux fois. S'il souhaite produire un message de plus, il doit choisir un r_i déjà utilisé (sinon la vérification échoue puisque le nouveau r_i ne sera pas dans l'accumulateur). On obtient donc 2 messages avec le même t_i. Les paramètres du CH des 2 messages sont nécessairement différents (sinon, les 2 messages sont identiques). On a d'après la construction du compteur :

$$y_1 = r_i - R_1 s$$

pour le premier message et :

$$y_2 = r_i - R_2 s$$

pour le second.

Puisque R_1 et R_2 sont différents, on peut calculer :

$$s = \frac{y_2 - y_1}{R_1 - R_2}$$

et on récupère ainsi s qui est l'information secrète du Rédacteur.

Remarque :

En cas de fraude on peut donc récupérer s que le Rédacteur doit ne pas vouloir rendre public. Seulement, s est choisi par le Rédacteur et le Signataire ne doit pas en connaître la valeur, sous peine de perdre l'identification de l'origine de la licence. Rien ne force le Rédacteur à mettre dans s quoique ce soit de sensible. Il peut tout à fait choisir pour s une valeur quelconque ne représentant absolument rien. Il ne risque donc rien de plus qu'avant la mise en place du compteur.

Plusieurs approches sont possibles :

– Tout d'abord, le compteur tel qu'il est présenté, facilite tout de même la détection des fraudes. En plus de l'identification de la source des copies distribuées, il n'est maintenant plus nécessaire de détecter qu'il y a trop de copies pour repérer la fraude, il suffit de trouver 2 messages ayant le même r_i.

– Ensuite, il est possible pour le Signataire de prendre v égale à une clé publique du Rédacteur, s est alors une clé privée. Ainsi en cas de fraude, la clé privée du Rédacteur se retrouve à la portée de tous. Cette méthode est très certainement excessive. En effet une fois s dévoilée, n'importe qui aura accès aux messages chiffrés avec cette clé.
– La dernière solution est sans doute plus modérée. Le Signataire utilise le v fourni par le Rédacteur comme une clé ElGamal. Le Rédacteur en possède donc la clé privée s. Avec v, le Signataire chiffre une information de son choix sur le Rédacteur et l'ajoute à la signature. Ainsi, si s est révélée, n'importe qui aura accès à cette information. Le couple de clés (v, s) ne sert qu'ici, cette solution est donc beaucoup moins brutale que la précédente.
  Cette méthode a également l'avantage de permettre au Signataire de choisir ce qu'il souhaite chiffrer. Il a donc la possibilité de choisir une information plus ou moins sensible en fonction du message et ainsi adapter la sanction en cas de fraude. De son côté, le Rédacteur a maintenant une raison valable de souhaiter garder s secret.

3.4.2 Résumé

Le compteur décrit précédemment a donc toutes les propriétés évoquées, il permet d'identifier l'origine des copies, il empêche le Rédacteur de dépasser le nombre de versions permises et enfin la vérification se fait de manière statique.

Une fois de plus, le Signataire peut pré calculer l'ensemble des témoins permettant de vérifier que les t_i sont bien dans l'accumulateur pour éviter que ce soit le Rédacteur qui fasse les calculs.

3.4.3 Conclusion

Nous avons apporté une solution pour tout les objectifs cités au début du chapitre. Nous allons voir maintenant comment ces solutions peuvent être adaptées à notre modèle de gestion de licences.

Chapitre 4

Application aux modèle DRM

Toutes les solution présentées précédemment s'adapte tout naturellement au modèle des licences.

Empêcher la transitivité des signatures permet d'empêcher qu'un utilisateur construise une licence avec des droits supplémentaires à ceux dont il est censé disposer. Les Accumulateurs sont idéaux pour s'assurer que les droits ont une valeur cohérente après modifications par le Rédacteur. Enfin le compteur permet d'empêcher le Rédacteur de distribuer le contenu à tout le monde court-circuitant ainsi le système DRM. Le fait de sanctionner le Rédacteur en cas de fraude prend ici tout son sens puisqu'en règle générale, les licences protègent des contenus payant.

4.1 Modèle final de licence

La structure de la licence, plus précisément de sa signature, a été modifiée par rapport au modèle d'origine. Après ajout du contrôle de modifications des droits et du compteur, sa construction et sa vérification ont par conséquent changé.

4.1.1 Construction

Le Rédacteur reçoit de la part du SL la trapdoor qui lui permet d'effectuer des modifications, il reçoit en même temps la première licence qui servira de base aux modifications :

$$L = [id_L, Id_c, m, CEK_c, S]$$

avec :

$$S = [Sign_{SL}(Id_L, Id_c, Hacc(\mathcal{M}), Hacc(\mathcal{T}), v, CH(m||CEK_c)), R]$$

$\mathcal{T}$ et v ont été fournis par le Rédacteur.

Le SL peut éventuellement lui envoyer aussi la liste des témoins pour les deux accumulateurs si le Rédacteur n'a pas les capacités de les construire lui-même.

Construction d'une nouvelle version de la licence :

1. Le rédacteur commence par déchiffrer la CEK puis il la rechiffre avec la clé publique du nouveau destinataire.
2. Il choisit les droits m_i de la nouvelle licence dans l'ensemble $\mathcal{M}$.
3. Il choisit l'un des t_j non encore utilisés.
4. Grâce à la trapdoor, il construit le nouveau paramètre R' du CH tel que :
$$CH(m||CEK_c, R) = CH(m_i||CEK_{c'}||t_j, R')$$
5. Il calcule les témoins pour m_i et t_i, z_m et z_t.
6. Il calcule le compteur $y = r_j - R's$.
7. Finalement il construit la nouvelle licence L' tel que :
$$L = [id_L, Id_c, m_i, CEK_{c'}, S']$$
avec :
$$S' = [Sign_{SL}(Id_L, Id_c, Hacc(\mathcal{M}), Hacc(\mathcal{T}), v, CH(m_i||CEK_{c'}||t_j)),$$
$$R', z_m, z_t, t_j, y, v, Hacc(\mathcal{M}), Hacc(\mathcal{T})]$$

4.1.2 Vérification

Lorsque l'utilisateur final reçoit la licence, en plus de vérifier que Id_c et m_i sont bien ceux attendus, il effectue les vérifications suivantes :

1. Il vérifie que t_i et m_i sont bien dans les accumulateurs en comparant :
$$Hacc(\mathcal{M}) \stackrel{?}{=} {z_m}^{H^*(m_i)}$$
et
$$Hacc(\mathcal{T}) \stackrel{?}{=} {z_t}^{H^*(t_j)}$$
2. Il vérifie que le compteur est correct :
$$t_j \stackrel{?}{=} v^{R'}\alpha^y$$

3. Il calcule le CH à l'aide de R' et vérifie que

$$Sign_{SL}(Id_L, Id_c, Hacc(\mathcal{M}), Hacc(\mathcal{T}), v, CH(m_i||CEK_{c'}||t_j))$$

est correcte.

Si une seule de ces étapes échoue, l'utilisateur considère la licence comme invalide. Sinon, il peut alors déchiffrer la CEK puis le contenu.

4.2 Conclusion

La résolution des problèmes dus à la construction du CH et l'ajout des nouvelles fonctionnalités à la licence c'est donc fait en faisant en sorte que la solution reste praticable. La taille de la licence augmente un peu mais reste raisonnable. Les calculs supplémentaires peuvent être quasiment nuls pour les Rédacteurs et les vérifieurs si le SL se charge de faire les pré calculs. Enfin le volume d'informations échangées n'augmente pas beaucoup, sauf si le SL fait les pré-calculs, mais dans ce cas, les communications n'ont pas besoin d'être sécurisées car les témoins des accumulateurs peuvent tout à fait être publics.

Chapitre 5

Maquette

Un autre but de mon stage, outre l'amélioration de la solution existante, était d'implementer en JAVA l'ensemble du modèle TSS. Le but final étant d'inclure cette signature dans une maquette déjà existante.

5.1 La maquette existante

La maquette permet l'échange de fichiers (musique, vidéos et photos) au sein d'un groupe s'apparentant à une famille. Certains membres du groupe peuvent également acquérir de nouveaux contenus auprès de divers fournisseurs. Les différents utilisateurs sont organisés en fonction de leur rôle dans la famille (parent, adolescent et enfant). Le groupe peut également accueillir des amis des différents membres de la famille. L'organisation du groupe est faite à l'aide d'un graphe de façon à pouvoir inclure une notion de hiérarchie. Ainsi une personne à un certain niveau aura accès aux données des personnes situées «en dessous» d'elle. Par exemple les parents ont accès aux données des adolescents, les adolescents à celles des enfants etc.

La maquette introduit un nouvelle forme de licence. Si l'on a toujours les licences du modèle DRM qui sont chargées de contrôler l'utilisation faite du contenu qu'elles accompagnent, on a maintenant une *métalicence* qui à pour rôle de contrôler les actions sur les licences elles mêmes. Initialement, la maquette gère la possibilité ou non de transférer un fichier à une seconde personne à l'aide d'une métalicence. L'utilisateur initial, si la métalicence l'y autorise, modifie alors la licence qui accompagne toujours le contenu (modification de la CEK) pour le nouvelle utilisateur. En raison de cette modification, les licences étaient dépourvues de signature. En effet comme l'utilisateur initial n'a aucun moyen de reconstruire une signature valable, le transfert aurait été impossible. Le but était donc d'utiliser le modèle de

signatures déléguées développé.

5.2 Développement effectué

Le développement c'est donc fait en JAVA, avec pour consigne d'éviter d'utiliser des librairies extérieures afin de limiter la taille de l'application mais également pour faciliter l'éventuel portage sur d'autres plates-formes.

En raison de la modification constante du modèle à développer tout au long du stage et que je découvrais le fonctionnement et l'utilisation des bases de données en JAVA, je n'ai pas pu aller jusqu'au bout des objectifs. Néanmoins, j'a eu le temps d'avancer le développement. Ainsi j'ai implémenté (à l'extérieur de la maquette) l'ensemble des fonctions nécessaires au modèle. Cela inclut donc l'implémentation des accumulateurs (construction, vérification), du compteur et de l'*id-based Chameleon Hash* (calcul, vérification et modification).

Le modèle est donc complètement implémenté et fonctionne. Cependant il est est pour l'instant à l'extérieur de la maquette.

5.3 Ce qu'il reste à faire

Comme expliqué précédemment, l'implémentation est faite à l'extérieur de la maquette. Il faut donc l'introduire à l'intérieur de celle ci. Il faut pour cela modifier l'étape de construction de la licence afin d'y ajouter la signature. Il faut ensuite pouvoir échanger la trappe entre le Signataire et le Rédacteur. Il faut que le Rédacteur puisse stocker, dans la métalicence par exemple, l'ensemble des r_i, l'ensemble $\mathcal{T}$ et la trapdoor de façon à pouvoir les récupérer si l'application est fermée puis relancée. Enfin, il faut maintenant vérifier la signature de la licence avant chaque accès au contenu.

Chapitre 6

Apports du stage et conclusion

Ce stage m'a permis de mettre en pratique un grand nombre de choses apprises au cours de mes années d'études. J'ai eu la chance de pouvoir au cours de mon stage aborder le sujet tant d'un point de vu théorique avec l'étude et l'amélioration d'une solution de signature et d'un point de vu technique lorsqu'il a fallut implémenter en JAVA ce qui avait été fait au cours de l'étape précédente.

J'ai beaucoup appris au cours de ce stage qui fut une expérience enrichissante en tout point. D'un point de vue personnel tout d'abord, travailler en équipe avec des personnes très expérimentées et compétentes peut être un peu intimidant au début. D'un point de vue professionnelle ensuite. C'est sans aucun doute grâce à mon stage autant qu'au reste de mes études que j'ai obtenu la thèse que je souhaitais, celle ci poursuivant idéalement le sujet de mon stage.

Il est surprenant de voir comment 5 mois de stage peuvent orienter de façon très différentes 5 années d'études.

Bibliographie

[1] G. Ateniese, D.H. Chou, B. De Medeiros, G. Tsudik. Sanitizable Signature. 2005

[2] Klonowski, Lauks. Extended Sanitizable Signatures. 2006

[3] R. Steinfeld, L. Bull, Y. Zheng. Content extraction signature. 2001

[4] G. Ateniese, B. De Medeiros. Identity-based Chameleon Hash and Applications. 2003

[5] H. Krawczyk, T. Rabin. Chameleon Signatures. 1996

[6] S. Canard, F. Laguillaumie, M. Milhau. Trapdoor Sanitizable Signatures and their application to content protection. 2008

[7] Rivest. Two Signatures Schemes.

[8] Schnorr. Efficient Identification and Signatures for Smart Cards. 1989

[9] Benaloh, De Mare. One-way accumulators : a decentralised alternative to digital signatures. 1994

[10] PKCS #1 v2.1 : RSA Cryptography Standard.

[11] Micali, Rivest. Transitive signature schemes. 2000

[12] Johnson, Molnar, Song, Wagner. Homomorphic signature schemes. 2002